In [22]:
```
import pandas as pd
df = pd.read_csv("C:\\Users\\USER\\Desktop\\machine learning ICA folder\\house_pric
dt = pd.read_csv("C:\\Users\\USER\\Desktop\\machine learning ICA folder\\house_pric
```

In [2]:
```
print(dt)
```

```
       SquareFeet  Bedrooms  Bathrooms Neighborhood  YearBuilt          Price
0            2126         4          1        Rural       1969  215355.283618
1            2459         3          2        Rural       1980  195014.221626
2            1860         2          1       Suburb       1970  306891.012076
3            2294         2          1        Urban       1996  206786.787153
4            2130         5          2       Suburb       2001  272436.239065
...           ...       ...        ...          ...        ...            ...
49995        1282         5          3        Rural       1975  100080.865895
49996        2854         2          2       Suburb       1988  374507.656727
49997        2979         5          3       Suburb       1962  384110.555590
49998        2596         5          2        Rural       1984  380512.685957
49999        1572         5          3        Rural       2011  221618.583218

[50000 rows x 6 columns]
```

In [23]:
```
# Examine the data type
print(dt.dtypes)
```

```
SquareFeet        int64
Bedrooms          int64
Bathrooms         int64
Neighborhood     object
YearBuilt         int64
Price           float64
dtype: object
```

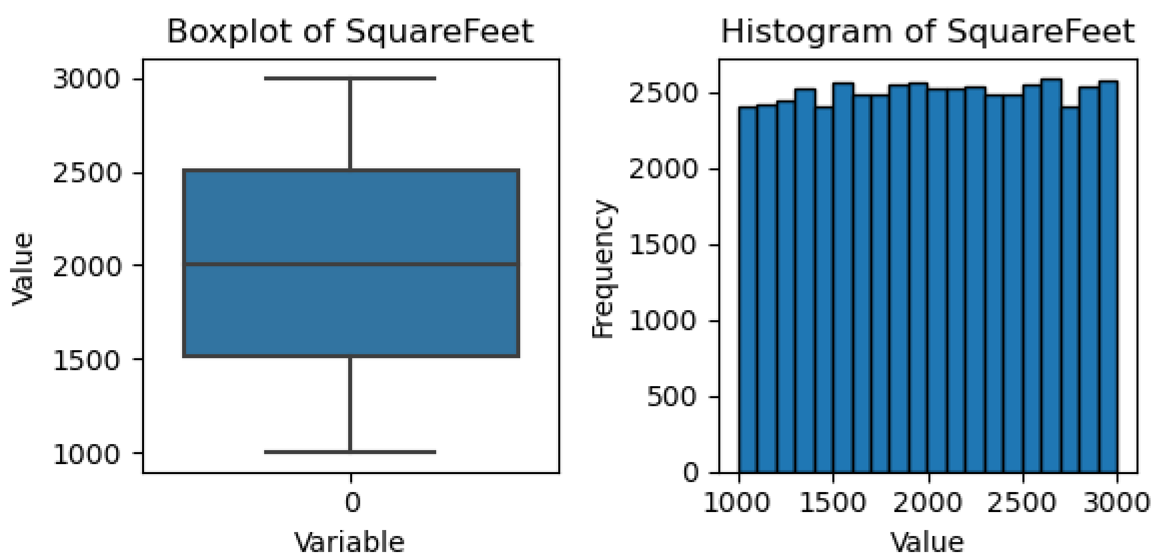
In [25]:
```
# Examine the dataset for duplicate row
duplicate_rows = dt[dt.duplicated()]

# Print the duplicate rows
print("Duplicate Rows:")
print(duplicate_rows)
```

```
Duplicate Rows:
Empty DataFrame
Columns: [SquareFeet, Bedrooms, Bathrooms, Neighborhood, YearBuilt, Price]
Index: []
```

In [26]:
```
# Examine the dataset for missing values
dt.isnull().sum()
```

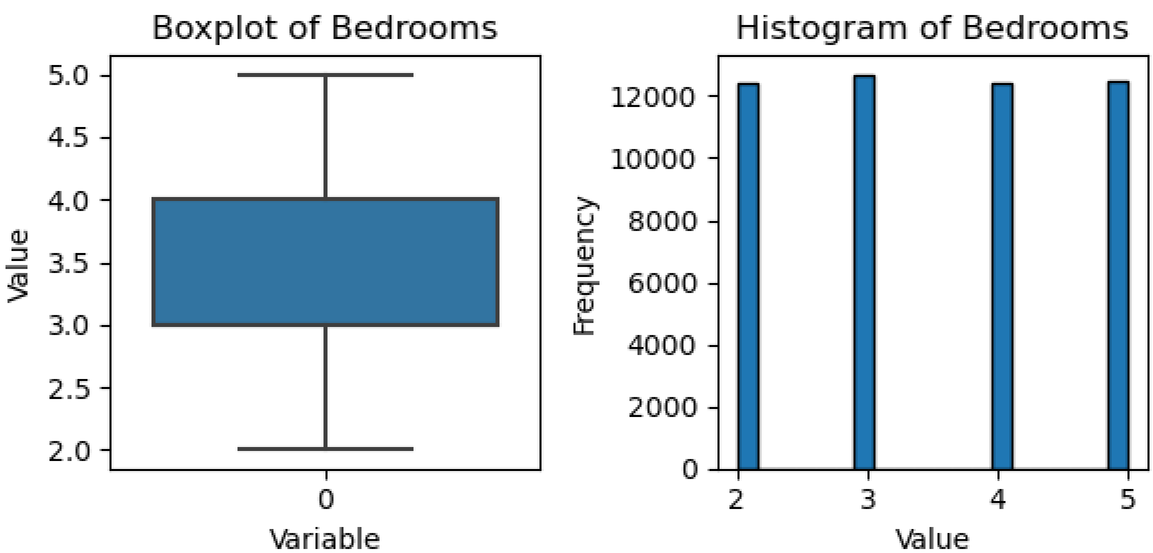
Out[26]:
```
SquareFeet      0
Bedrooms        0
Bathrooms       0
Neighborhood    0
YearBuilt       0
Price           0
dtype: int64
```

In [27]:
```
# Examine each  numerical variable for outliers using box plot and histogram

#Squarefeet column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['SquareFeet'], ax=axs[0])
axs[0].set_title('Boxplot of SquareFeet')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['SquareFeet'], bins=20, edgecolor='k')  # Adjust the number of bins
axs[1].set_title('Histogram of SquareFeet')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```



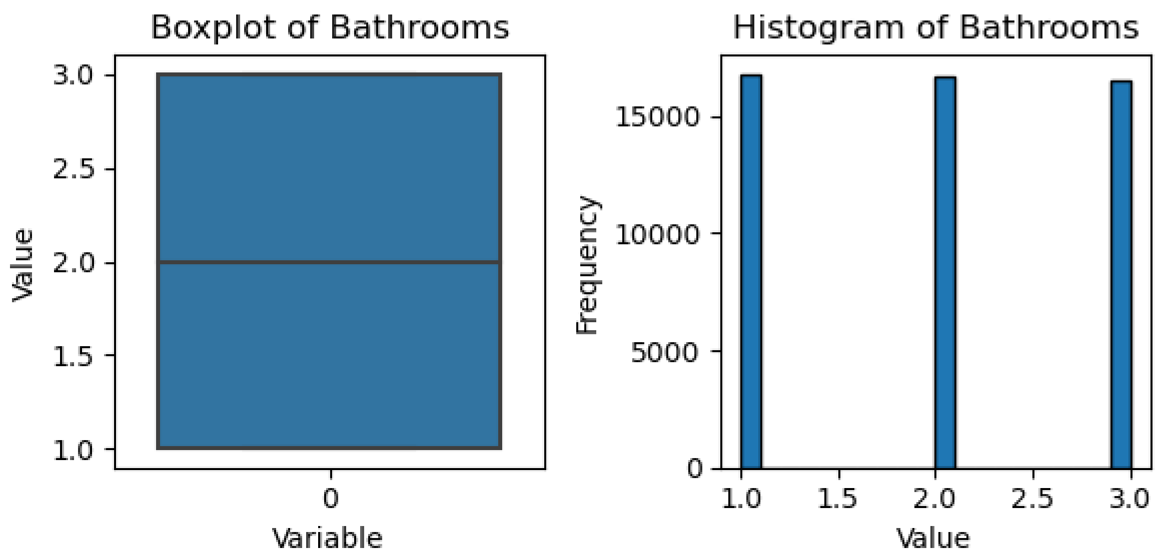
In [28]:
```
#Bedrooms column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['Bedrooms'], ax=axs[0])
axs[0].set_title('Boxplot of Bedrooms')
```

```
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['Bedrooms'], bins=20, edgecolor='k')  # Adjust the number of bins as
axs[1].set_title('Histogram of Bedrooms')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```

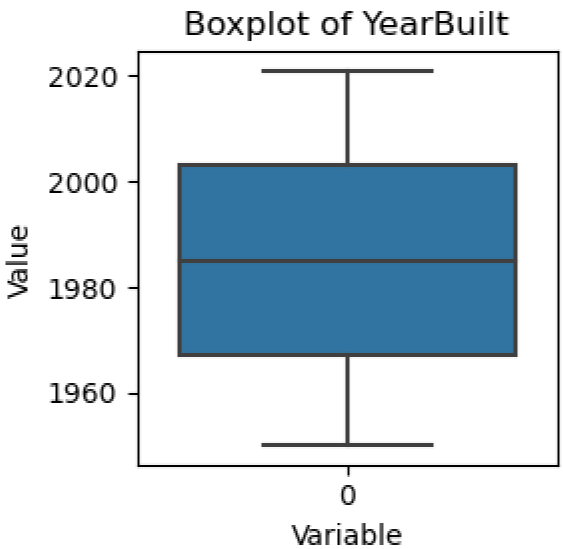


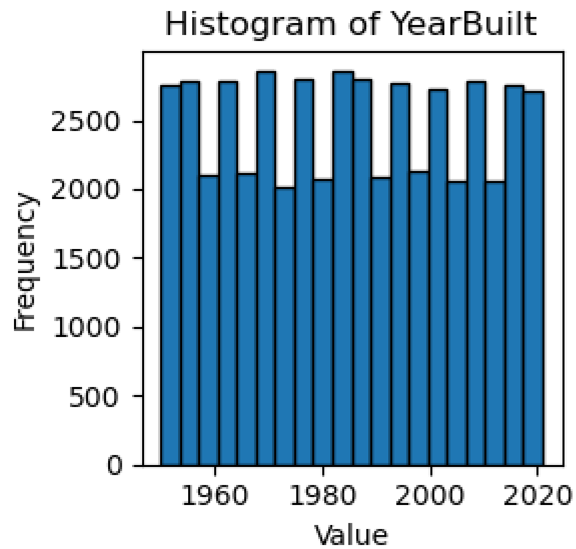
In [29]:
```
#Bathrooms column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['Bathrooms'], ax=axs[0])
axs[0].set_title('Boxplot of Bathrooms')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['Bathrooms'], bins=20, edgecolor='k')  # Adjust the number of bins a
axs[1].set_title('Histogram of Bathrooms')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()
```

```
# Show the plots
plt.show()
```

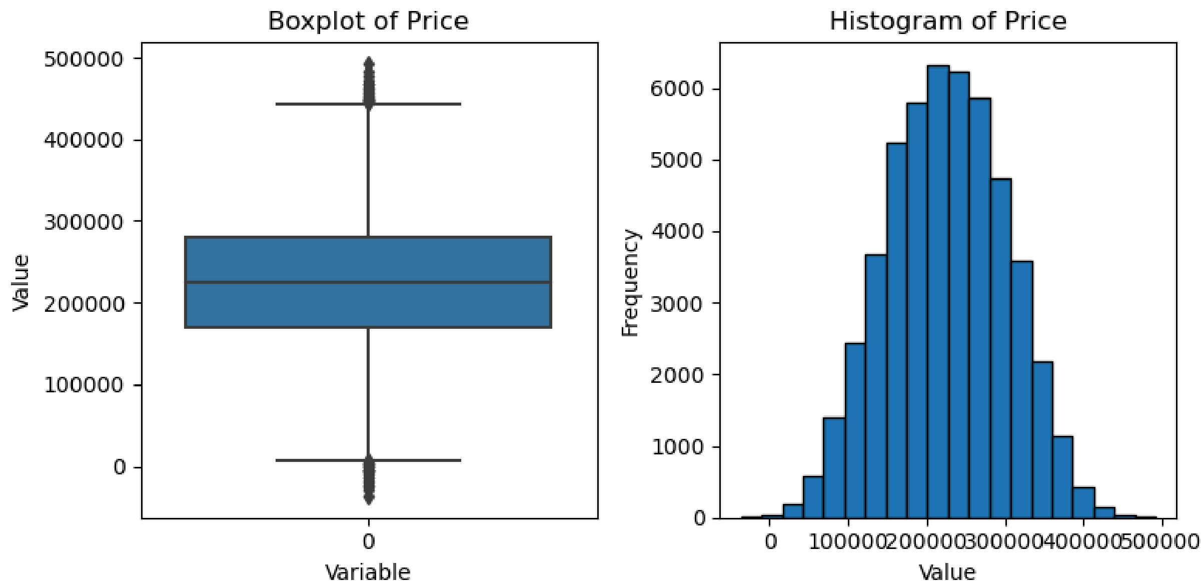


In [30]:
```
#YearBuilt column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['YearBuilt'], ax=axs[0])
axs[0].set_title('Boxplot of YearBuilt')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['YearBuilt'], bins=20, edgecolor='k')  # Adjust the number of bins a
axs[1].set_title('Histogram of YearBuilt')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```



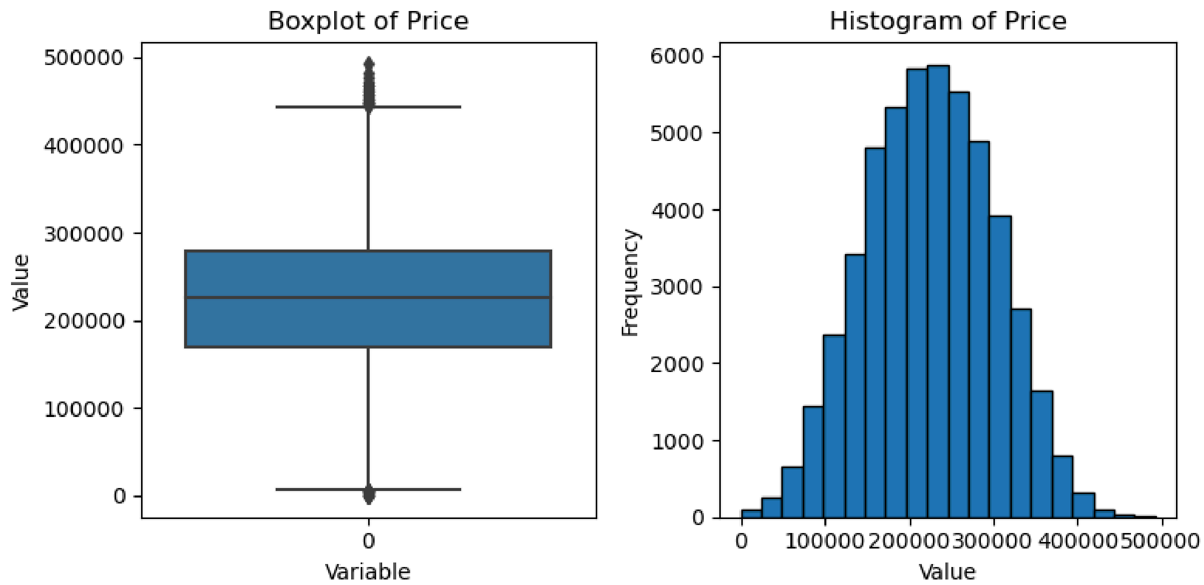


In [31]:
```
#Price column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['Price'], ax=axs[0])
axs[0].set_title('Boxplot of Price')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['Price'], bins=20, edgecolor='k')  # Adjust the number of bins as ne
axs[1].set_title('Histogram of Price')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```



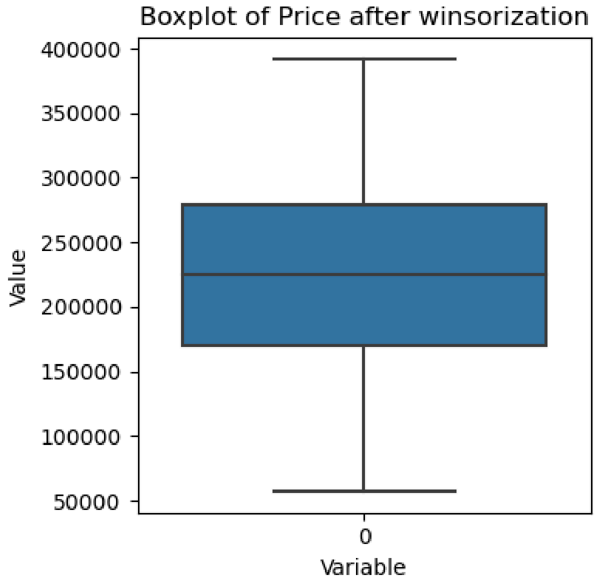
In [32]:
```
#there are outliers in the price column
#from the histogram, the bins extended beyound 0,this implies negative values
#from domian knowledge, price of a house can not be negative
#handle the negative values by taking the absolute value of the price
dt['Price'] = dt['Price'].abs()
```

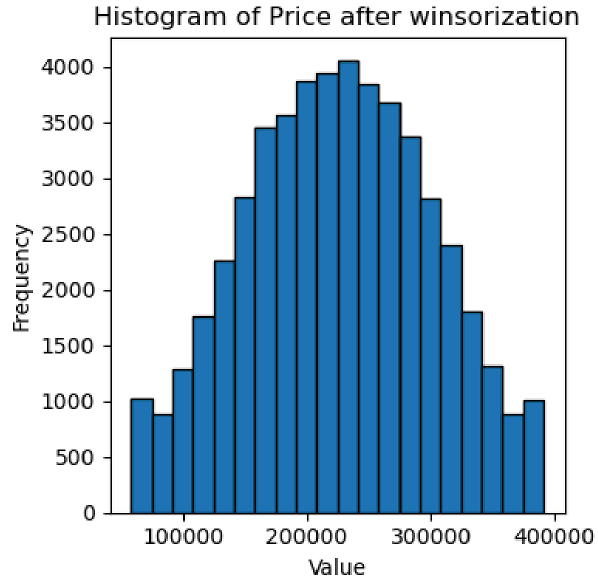
In [33]:
```
#confirm if the absolute vlues have been captured using box plot and histogram
#Price column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['Price'], ax=axs[0])
axs[0].set_title('Boxplot of Price after taking the absolute value')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['Price'], bins=20, edgecolor='k')  # Adjust the number of bins as ne
axs[1].set_title('Histogram of Price after taking the absolute value')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```



In [35]:
```
# handle the remaining outliers using wisorization method
import numpy as np


# Set the threshold (e.g., 1st and 99th percentiles)
lower_threshold = np.percentile(dt['Price'], 1)
upper_threshold = np.percentile(dt['Price'], 99)

# Replace outliers with threshold values
dt['Price'] = np.where(dt['Price'] < lower_threshold, lower_threshold, dt['Price'])
dt['Price'] = np.where(dt['Price'] > upper_threshold, upper_threshold, dt['Price'])
```

In [37]:
```
# confirm if the wisorization has handled the outliers
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=dt['Price'], ax=axs[0])
axs[0].set_title('Boxplot of Price after winsorization')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(dt['Price'], bins=20, edgecolor='k')  # Adjust the number of bins as ne
axs[1].set_title('Histogram of Price after winsorization')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```





In [40]:
```
# Exploratory data analysis(EDA)
#perform exploratory data analysis using correlation matrics
print(dt)
```

```
       SquareFeet  Bedrooms  Bathrooms Neighborhood  YearBuilt          Price
0            2126         4          1        Rural       1969  215355.283618
1            2459         3          2        Rural       1980  195014.221626
2            1860         2          1       Suburb       1970  306891.012076
3            2294         2          1        Urban       1996  206786.787153
4            2130         5          2       Suburb       2001  272436.239065
...           ...       ...        ...          ...        ...            ...
49995        1282         5          3        Rural       1975  100080.865895
49996        2854         2          2       Suburb       1988  374507.656727
49997        2979         5          3       Suburb       1962  384110.555590
49998        2596         5          2        Rural       1984  380512.685957
49999        1572         5          3        Rural       2011  221618.583218

[50000 rows x 6 columns]
```

In [41]:
```
# EDA purposes, label  encode neighborhood column
dt['Neighborhood'].value_counts()
```

Out[41]:
```
Neighborhood
Suburb    16721
Rural     16676
Urban     16603
Name: count, dtype: int64
```

In [42]:
```
#Convert “Suburb” to 0, “Rural” to 1 and "Urban" to 2 using the Python code below
dt["Neighborhood"].replace({'Suburb': 0, 'Rural': 1, 'Urban': 2}, inplace=True)
```

In [43]:
```
#print dt to confirm if the label encoding has been captured
print(dt)
```

```
       SquareFeet  Bedrooms  Bathrooms  Neighborhood  YearBuilt          Price
0            2126         4          1             1       1969  215355.283618
1            2459         3          2             1       1980  195014.221626
2            1860         2          1             0       1970  306891.012076
3            2294         2          1             2       1996  206786.787153
4            2130         5          2             0       2001  272436.239065
...           ...       ...        ...           ...        ...            ...
49995        1282         5          3             1       1975  100080.865895
49996        2854         2          2             0       1988  374507.656727
49997        2979         5          3             0       1962  384110.555590
49998        2596         5          2             1       1984  380512.685957
49999        1572         5          3             1       2011  221618.583218

[50000 rows x 6 columns]
```

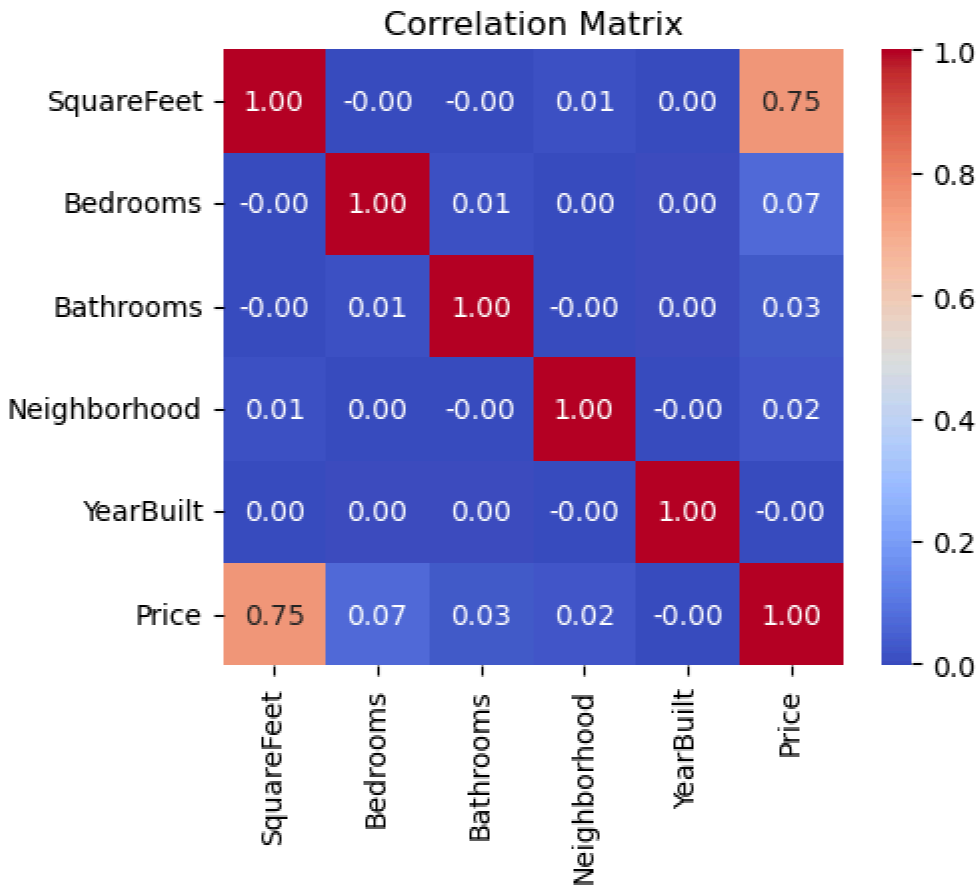
In [44]:

```python
# calculate the correlation coefficeint
import pandas as pd

# Assuming df is your DataFrame
correlation_matrix = dt.corr()

# Print the correlation matrix
print(correlation_matrix)
```

```
              SquareFeet  Bedrooms  Bathrooms  Neighborhood  YearBuilt  \
SquareFeet      1.000000 -0.002638  -0.003275      0.012234   0.000482
Bedrooms       -0.002638  1.000000   0.007405      0.000523   0.003147
Bathrooms      -0.003275  0.007405   1.000000     -0.003139   0.003748
Neighborhood    0.012234  0.000523  -0.003139      1.000000  -0.003375
YearBuilt       0.000482  0.003147   0.003748     -0.003375   1.000000
Price           0.752872  0.072462   0.028068      0.021482  -0.002118

                 Price
SquareFeet    0.752872
Bedrooms      0.072462
Bathrooms     0.028068
Neighborhood  0.021482
YearBuilt    -0.002118
Price         1.000000
```

In [45]:

```python
# plot the corelation matrix chart using heat map
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming dt is your DataFrame
correlation_matrix = dt.corr()

# Plot correlation matrix heatmap
plt.figure(figsize=(5, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



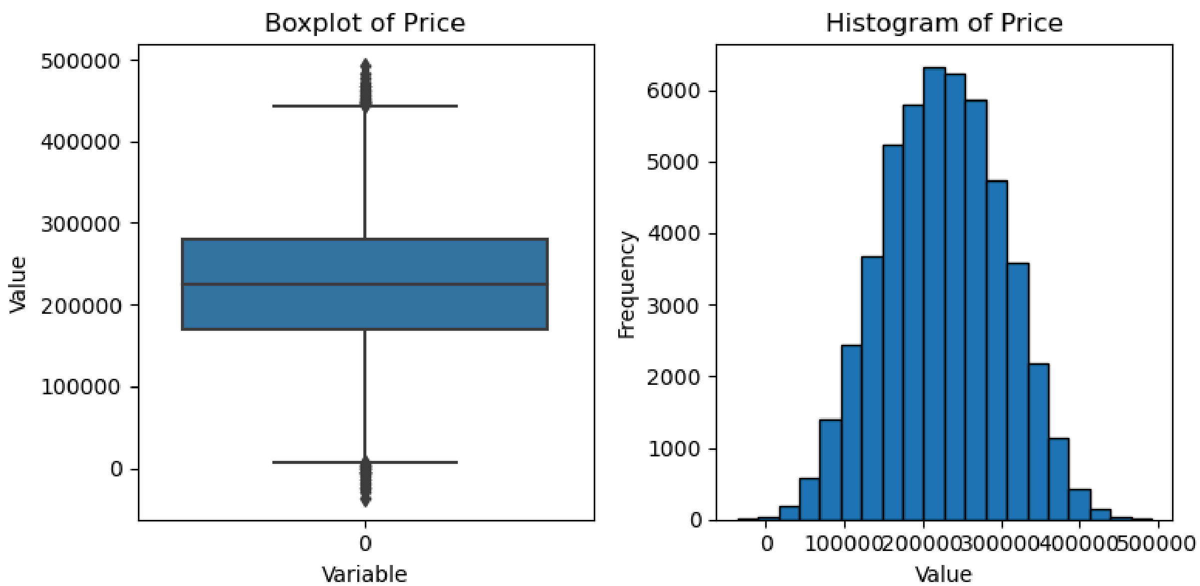
In [3]:
```
#having examined the linear relationship among the variables using correlation plot
# (df) using the same step, but apply one hot encoding on the neighborhood column t
#for my analysis.
import pandas as pd
df = pd.read_csv("C:\\Users\\USER\\Desktop\\machine learning ICA folder\\house_pric
```

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

```
       SquareFeet  Bedrooms  Bathrooms Neighborhood  YearBuilt          Price
0            2126         4          1        Rural       1969  215355.283618
1            2459         3          2        Rural       1980  195014.221626
2            1860         2          1       Suburb       1970  306891.012076
3            2294         2          1        Urban       1996  206786.787153
4            2130         5          2       Suburb       2001  272436.239065
...           ...       ...        ...          ...        ...            ...
49995        1282         5          3        Rural       1975  100080.865895
49996        2854         2          2       Suburb       1988  374507.656727
49997        2979         5          3       Suburb       1962  384110.555590
49998        2596         5          2        Rural       1984  380512.685957
49999        1572         5          3        Rural       2011  221618.583218

[50000 rows x 6 columns]
```

In [5]:
```
# clean Price column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
```

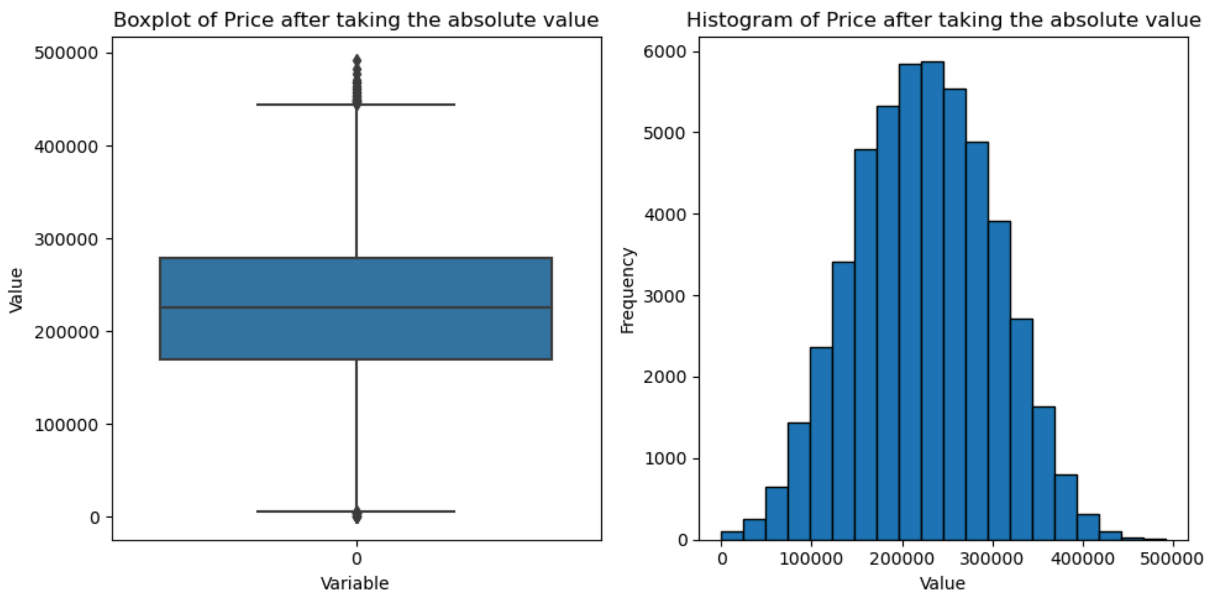
```
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['Price'], ax=axs[0])
axs[0].set_title('Boxplot of Price')
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['Price'], bins=20, edgecolor='k')  # Adjust the number of bins as ne
axs[1].set_title('Histogram of Price')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```



In [6]:
```
#there are outliers in the price column
#from the histogram, the bins extended beyound 0,this implies negative values
#from domian knowledge, price of a house can not be negative
#handle the negative values by taking the absolute value of the price
df['Price'] = df['Price'].abs()
```

In [7]:
```
#confirm if the absolute vlues have been captured using box plot and histogram
#Price column
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(10, 5))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['Price'], ax=axs[0])
axs[0].set_title('Boxplot of Price after taking the absolute value')
```

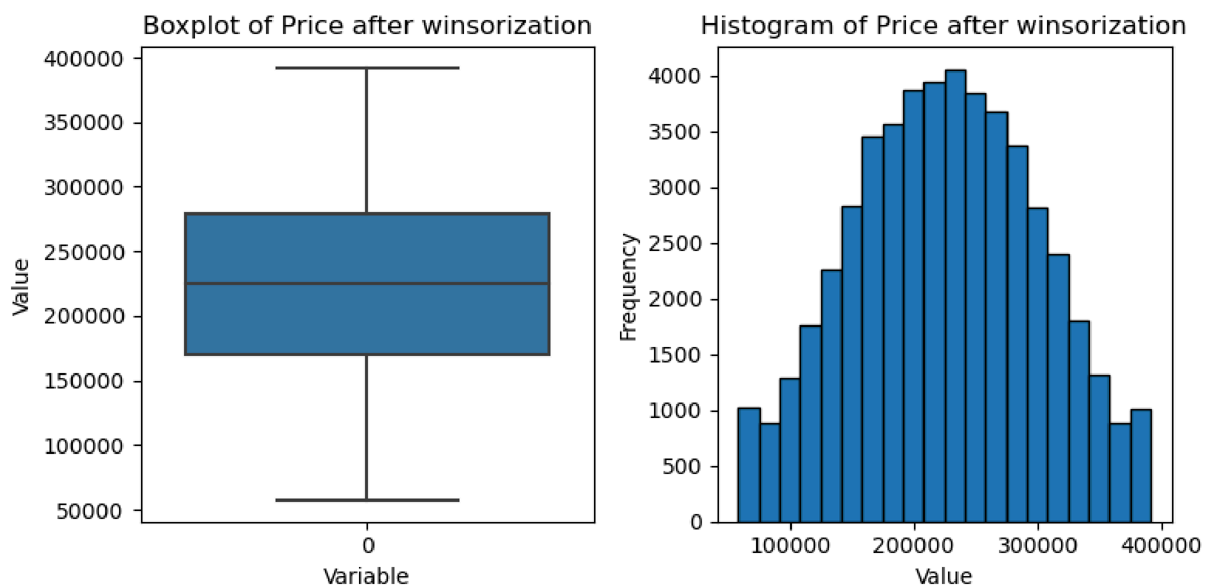
```
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['Price'], bins=20, edgecolor='k')  # Adjust the number of bins as ne
axs[1].set_title('Histogram of Price after taking the absolute value')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```



In [8]:
```
# handle the remaining outliers using wisorization method
import numpy as np


# Set the threshold (e.g., 1st and 99th percentiles)
lower_threshold = np.percentile(df['Price'], 1)
upper_threshold = np.percentile(df['Price'], 99)

# Replace outliers with threshold values
df['Price'] = np.where(df['Price'] < lower_threshold, lower_threshold, df['Price'])
df['Price'] = np.where(df['Price'] > upper_threshold, upper_threshold, df['Price'])
```

In [9]:
```
# confirm if the wisorization has handled the outliers
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(8, 4))  # Adjust figsize as needed

# Plot the boxplot on the first subplot
sns.boxplot(data=df['Price'], ax=axs[0])
axs[0].set_title('Boxplot of Price after winsorization')
```

```
axs[0].set_ylabel('Value')
axs[0].set_xlabel('Variable')

# Plot the histogram on the second subplot
axs[1].hist(df['Price'], bins=20, edgecolor='k')  # Adjust the number of bins as ne
axs[1].set_title('Histogram of Price after winsorization')
axs[1].set_xlabel('Value')
axs[1].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plots
plt.show()
```



In [10]:
```
# future enginering analysis
print(df)
```

```
       SquareFeet  Bedrooms  Bathrooms Neighborhood  YearBuilt          Price
0            2126         4          1        Rural       1969  215355.283618
1            2459         3          2        Rural       1980  195014.221626
2            1860         2          1       Suburb       1970  306891.012076
3            2294         2          1        Urban       1996  206786.787153
4            2130         5          2       Suburb       2001  272436.239065
...           ...       ...        ...          ...        ...            ...
49995        1282         5          3        Rural       1975  100080.865895
49996        2854         2          2       Suburb       1988  374507.656727
49997        2979         5          3       Suburb       1962  384110.555590
49998        2596         5          2        Rural       1984  380512.685957
49999        1572         5          3        Rural       2011  221618.583218

[50000 rows x 6 columns]
```

In [11]:
```
# one hot encode the neighborhood column to create binary columns of its features.
# Use get_dummies to one-hot encode the 'Neighborhood' column
df = pd.get_dummies(df, columns=['Neighborhood'])

# Convert True/False to 1/0
```

```
df = df.astype(int)

# Display the updated DataFrame
print(df)
```

```
       SquareFeet  Bedrooms  Bathrooms  YearBuilt   Price  Neighborhood_Rural  \
0            2126         4          1       1969  215355                   1
1            2459         3          2       1980  195014                   1
2            1860         2          1       1970  306891                   0
3            2294         2          1       1996  206786                   0
4            2130         5          2       2001  272436                   0
...           ...       ...        ...        ...     ...                 ...
49995        1282         5          3       1975  100080                   1
49996        2854         2          2       1988  374507                   0
49997        2979         5          3       1962  384110                   0
49998        2596         5          2       1984  380512                   1
49999        1572         5          3       2011  221618                   1

       Neighborhood_Suburb  Neighborhood_Urban
0                        0                   0
1                        0                   0
2                        1                   0
3                        0                   1
4                        1                   0
...                    ...                 ...
49995                    0                   0
49996                    1                   0
49997                    1                   0
49998                    0                   0
49999                    0                   0

[50000 rows x 8 columns]
```

In [12]:

```
# randonly reduce the numbers of the dataset rows from 50,000 to 10,000 for fast an
import pandas as pd

# Randomly sample 10,000 rows from the DataFrame
df_sampled = df.sample(n=10000, random_state=42)  # Setting random_state for reprod

# Display the shape of the sampled DataFrame to confirm the number of rows
print(df_sampled.shape)

# Optionally, you can reset the index of the sampled DataFrame
df_sampled.reset_index(drop=True, inplace=True)

# Display the sampled DataFrame
print(df_sampled)
```

```
(10000, 8)
      SquareFeet  Bedrooms  Bathrooms  YearBuilt   Price  Neighborhood_Rural  \
0           1894         5          1       1975  170835                   1
1           1001         5          3       1963  126913                   0
2           2264         4          3       1964  246611                   0
3           2299         5          1       1999  244250                   0
4           2651         2          1       1951  271127                   0
...          ...       ...        ...        ...     ...                 ...
9995        2005         3          3       1966  199265                   0
9996        1725         4          2       1960  241869                   0
9997        2885         3          2       1980  352184                   0
9998        1674         5          2       1967  244830                   0
9999        2229         3          2       1989  246512                   1

      Neighborhood_Suburb  Neighborhood_Urban
0                       0                   0
1                       1                   0
2                       1                   0
3                       1                   0
4                       1                   0
...                   ...                 ...
9995                    0                   1
9996                    1                   0
9997                    0                   1
9998                    0                   1
9999                    0                   0

[10000 rows x 8 columns]
```

In [13]:

```
# save the processed df on a new csv file for training analysis
import os
import pandas as pd


# Save the cleaned DataFrame to a new CSV file
# Replace 'cleaned_data.csv' with your desired file name
# df_cleaned.to_csv('cleaned_data.csv', index=False)

# Define df_cleaned by performing some cleaning/preprocessing steps
# For demonstration, let's say we're dropping rows with missing values
df_sampled_cleaned = df_sampled.dropna()

# Save the cleaned DataFrame to a new CSV file
df_sampled_cleaned.to_csv('sampled_cleaned_data.csv', index=False)

# Get the current working directory
current_directory = os.getcwd()

# Print the current directory
print("Current Directory:", current_directory)

# You can also use the os.listdir() function to list all files in the directory
print("Files in Current Directory:", os.listdir(current_directory))
```

```
Current Directory: C:\Users\USER
Files in Current Directory: ['.anaconda', '.cache.sqlite', '.conda', '.condarc', '.c
ontinuum', '.idlerc', '.ipynb_checkpoints', '.ipython', '.jupyter', '.matplotlib',
'.ms-ad', '.spyder-py3', '.vscode', 'AI.ICA.T1.ipynb', 'AI.ICA.TRIAL', 'AI.ICA.W.ipy
nb', 'anaconda3', 'AppData', 'Application Data', 'CIS4044-N-SDI-OPENMETEO-PARTIAL.d
b', 'cleaned_data.csv', 'Contacts', 'Cookies', 'Desktop', 'Documents', 'Downloads',
'Favorites', 'IntelGraphicsProfiles', 'LB.WK1.ipynb', 'Links', 'loan prediction.ipyn
b', 'Local Settings', 'Microsoft', 'ML. ICA. TRIAL1', 'ML.ICA.T1.ipynb', 'ML.ICA.T2.
ipynb', 'ML.ICA.W.ipynb', 'ML.ICA.WORK', 'Music', 'My Documents', 'my_new_director
y', 'NetHood', 'NTUSER.DAT', 'ntuser.dat.LOG1', 'ntuser.dat.LOG2', 'NTUSER.DAT{a2332
f17-cdbf-11ec-8680-002248483d79}.TxR.0.regtrans-ms', 'NTUSER.DAT{a2332f17-cdbf-11ec-
8680-002248483d79}.TxR.1.regtrans-ms', 'NTUSER.DAT{a2332f17-cdbf-11ec-8680-002248483
d79}.TxR.2.regtrans-ms', 'NTUSER.DAT{a2332f17-cdbf-11ec-8680-002248483d79}.TxR.blf',
'NTUSER.DAT{a2332f18-cdbf-11ec-8680-002248483d79}.TM.blf', 'NTUSER.DAT{a2332f18-cdbf
-11ec-8680-002248483d79}.TMContainer00000000000000000001.regtrans-ms', 'NTUSER.DAT{a
2332f18-cdbf-11ec-8680-002248483d79}.TMContainer00000000000000000002.regtrans-ms',
'ntuser.ini', 'OneDrive', 'Pictures', 'PrintHood', 'Recent', 'sampled1_cleaned_data.
csv', 'sampled_cleaned_data.csv', 'sampled_data.csv', 'Saved Games', 'ScStore', 'Sea
rches', 'SendTo', 'Start Menu', 'Templates', 'Untitled.ipynb', 'untitled.txt', 'Unti
tled1.ipynb', 'Untitled10.ipynb', 'Untitled11.ipynb', 'Untitled12.ipynb', 'Untitled1
3.ipynb', 'Untitled14.ipynb', 'Untitled2.ipynb', 'Untitled3.ipynb', 'Untitled4.ipyn
b', 'Untitled5.ipynb', 'Untitled6.ipynb', 'Untitled7.ipynb', 'Untitled8.ipynb', 'Unt
itled9.ipynb', 'Videos', 'your_database_file.db', 'your_database_path.db']
```

In [14]:
```python
# Concatenate the current directory with the file name
file_path = os.path.join(current_directory, 'sampled_cleaned_data.csv')

# Print the full file path
print("Full File Path:", file_path)
```

```
Full File Path: C:\Users\USER\sampled_cleaned_data.csv
```

In [15]:
```python
#import the cleaned df for analysis
#print its first five rows for comfirmation
import pandas as pd

# Full file path to the cleaned CSV file
file_path = "C:\\Users\\USER\\sampled_cleaned_data.csv"

# Load the cleaned dataset into a DataFrame
sampled_cleaned_df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame to verify it's loaded correctly
print(sampled_cleaned_df.head())
```

```
   SquareFeet  Bedrooms  Bathrooms  YearBuilt   Price  Neighborhood_Rural  \
0        1894         5          1       1975  170835                   1
1        1001         5          3       1963  126913                   0
2        2264         4          3       1964  246611                   0
3        2299         5          1       1999  244250                   0
4        2651         2          1       1951  271127                   0

   Neighborhood_Suburb  Neighborhood_Urban
0                    0                   0
1                    1                   0
2                    1                   0
3                    1                   0
4                    1                   0
```

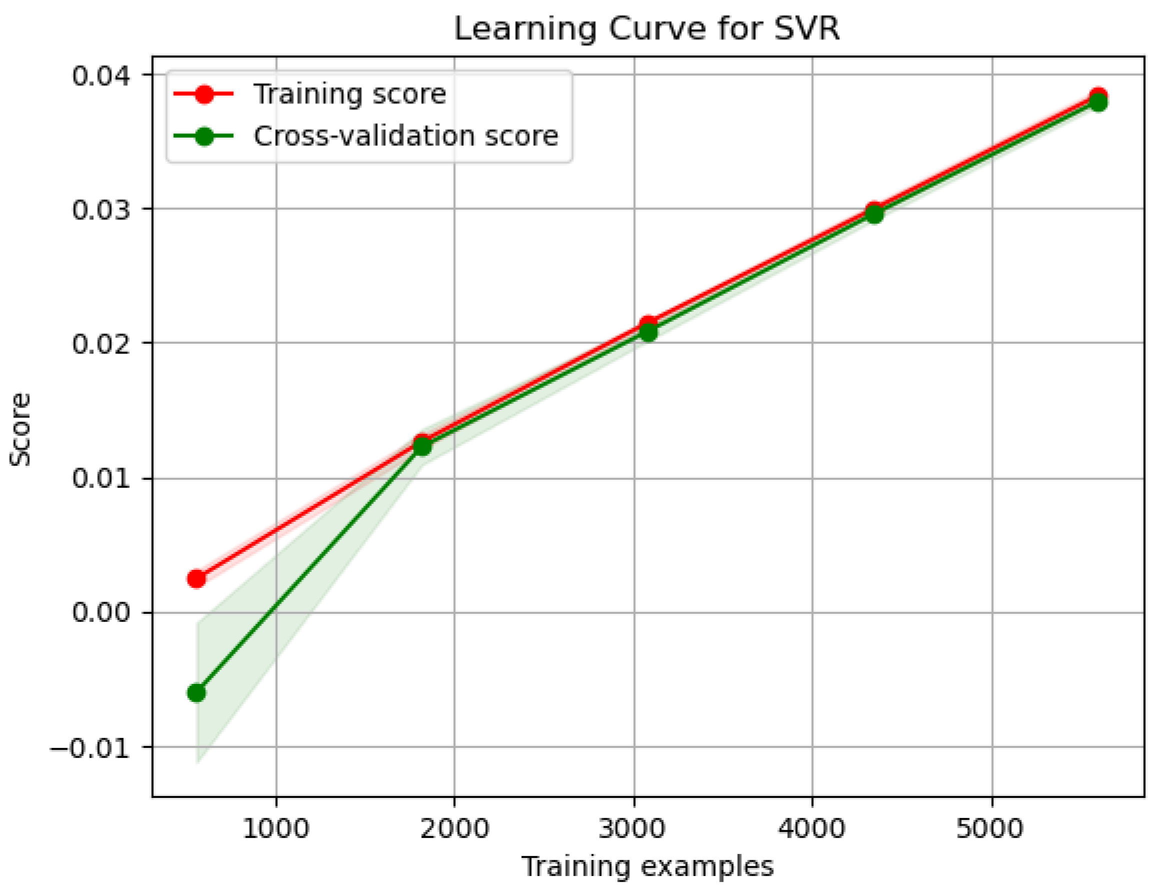
In [20]:

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, learning
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load your dataset
df = pd.read_csv("C:\\Users\\USER\\sampled_cleaned_data.csv")

# Extract features (X) and target variable (y)
X = df.drop('Price', axis=1)
y = df['Price']

# Step 1: Data Preprocessing
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 2: Train-Validation-Test Split
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, ran
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

# Step 3: Hyperparameter Tuning
param_grid = {
    'kernel': ['rbf'],
    'C': [0.1, 1, 10],
    'epsilon': [0.1, 0.2, 0.5],
    'gamma': ['scale', 'auto', 0.1, 1, 10]
}
kfold = KFold(n_splits=5)
grid_search = GridSearchCV(SVR(), param_grid, cv=kfold, scoring='neg_mean_squared_e
grid_search.fit(X_train, y_train)

# Get best parameters
best_params_svr = grid_search.best_params_
best_svr = grid_search.best_estimator_
print("Best parameters of SVR:", best_params_svr)

# Step 4: Model Evaluation
y_val_pred = best_svr.predict(X_val)
```

```
mae_val = mean_absolute_error(y_val, y_val_pred)
mse_val = mean_squared_error(y_val, y_val_pred)
r2_val = r2_score(y_val, y_val_pred)
print("Validation Set - Mean Absolute Error (MAE):", mae_val)
print("Validation Set - Mean Squared Error (MSE):", mse_val)
print("Validation Set - Coefficient of Determination (R^2):", r2_val)

# Cross-validation metrics
cv_mae_scores = []
cv_mse_scores = []
cv_r2_scores = []
for train_index, val_index in kfold.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]
    best_svr.fit(X_train_fold, y_train_fold)
    y_val_pred_fold = best_svr.predict(X_val_fold)
    fold_mae = mean_absolute_error(y_val_fold, y_val_pred_fold)
    fold_mse = mean_squared_error(y_val_fold, y_val_pred_fold)
    fold_r2 = r2_score(y_val_fold, y_val_pred_fold)
    cv_mae_scores.append(fold_mae)
    cv_mse_scores.append(fold_mse)
    cv_r2_scores.append(fold_r2)
mean_cv_mae = np.mean(cv_mae_scores)
mean_cv_mse = np.mean(cv_mse_scores)
mean_cv_r2 = np.mean(cv_r2_scores)
print("Cross-Validation Mean Absolute Error (MAE): {:.2f}".format(mean_cv_mae))
print("Cross-Validation Mean Squared Error (MSE): {:.2f}".format(mean_cv_mse))
print("Cross-Validation Coefficient of Determination (R^2): {:.2f}".format(mean_cv_

# Predict on training set
y_train_pred = best_svr.predict(X_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)
print("Training Set - Mean Absolute Error (MAE):", mae_train)
print("Training Set - Mean Squared Error (MSE):", mse_train)
print("Training Set - Coefficient of Determination (R^2):", r2_train)

# Plot learning curve
train_sizes, train_scores, test_scores = learning_curve(best_svr, X_train, y_train,
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.title("Learning Curve for SVR")
plt.xlabel("Training examples")
plt.ylabel("Score")
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1,
                 color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
```

```
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
         label="Cross-validation score")
plt.legend(loc="best")

plt.show()

# Step 5: Final Evaluation on Test Set
y_test_pred = best_svr.predict(X_test)
mae_test = mean_absolute_error(y_test, y_test_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)
print("Test Set - Mean Absolute Error (MAE):", mae_test)
print("Test Set - Mean Squared Error (MSE):", mse_test)
print("Test Set - Coefficient of Determination (R^2):", r2_test)
```

```
Best parameters of SVR: {'C': 10, 'epsilon': 0.5, 'gamma': 0.1, 'kernel': 'rbf'}
Validation Set - Mean Absolute Error (MAE): 57816.73395095728
Validation Set - Mean Squared Error (MSE): 5030431499.21489
Validation Set - Coefficient of Determination (R^2): 0.043231267471325
Cross-Validation Mean Absolute Error (MAE): 60554.61
Cross-Validation Mean Squared Error (MSE): 5472507029.82
Cross-Validation Coefficient of Determination (R^2): 0.04
Training Set - Mean Absolute Error (MAE): 60524.881030581644
Training Set - Mean Squared Error (MSE): 5469579760.139359
Training Set - Coefficient of Determination (R^2): 0.0386621793616615
```



```
Test Set - Mean Absolute Error (MAE): 60284.88317326661
Test Set - Mean Squared Error (MSE): 5324250823.44652
Test Set - Coefficient of Determination (R^2): 0.040310127101546134
```

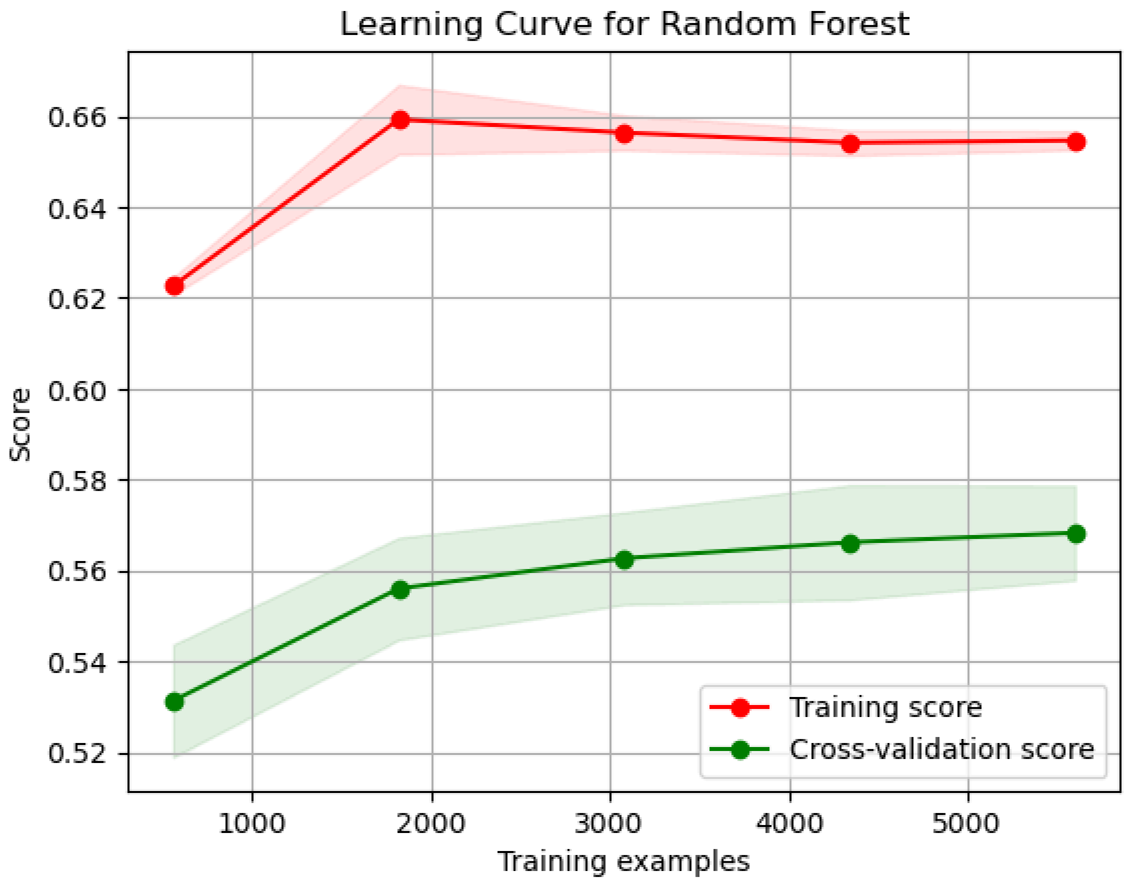
In [21]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, learning
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the dataset
df = pd.read_csv("C:\\Users\\USER\\sampled_cleaned_data.csv")

# Extract features (X) and target variable (y)
X = df.drop('Price', axis=1)
y = df['Price']

# Split dataset into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

# Define hyperparameter grid with wider ranges
param_grid = {
    'n_estimators': [50,100, 200, 300],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 8],
    'max_features': ['sqrt', 'log2'],
    'bootstrap': [True]
}

# Define k-fold cross-validation
kfold = KFold(n_splits=5)

# Initialize RandomForestRegressor
rf = RandomForestRegressor(max_features='auto', bootstrap=True, random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(rf, param_grid, cv=kfold, scoring='neg_mean_squared_erro

# Fit grid search to the data
grid_search.fit(X_train, y_train)

# Get best parameters
best_params = grid_search.best_params_
best_rf = grid_search.best_estimator_

# Print best parameters
print("Best Parameters of Random Forest:", best_params)

# Predict on validation set
y_val_pred = best_rf.predict(X_val)

# Model evaluation metrics on validation set
mae_val = mean_absolute_error(y_val, y_val_pred)
```

```
mse_val = mean_squared_error(y_val, y_val_pred)
r2_val = r2_score(y_val, y_val_pred)

print("Validation Set - Mean Absolute Error (MAE):", mae_val)
print("Validation Set - Mean Squared Error (MSE):", mse_val)
print("Validation Set - Coefficient of Determination (R^2):", r2_val)

# Cross-validation metrics
cv_mae_scores = []
cv_mse_scores = []
cv_r2_scores = []
for train_index, val_index in kfold.split(X_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]
    best_rf.fit(X_train_fold, y_train_fold)
    y_val_pred_fold = best_rf.predict(X_val_fold)
    fold_mae = mean_absolute_error(y_val_fold, y_val_pred_fold)
    fold_mse = mean_squared_error(y_val_fold, y_val_pred_fold)
    fold_r2 = r2_score(y_val_fold, y_val_pred_fold)
    cv_mae_scores.append(fold_mae)
    cv_mse_scores.append(fold_mse)
    cv_r2_scores.append(fold_r2)
mean_cv_mae = np.mean(cv_mae_scores)
mean_cv_mse = np.mean(cv_mse_scores)
mean_cv_r2 = np.mean(cv_r2_scores)
print("Cross-Validation Mean Absolute Error (MAE): {:.2f}".format(mean_cv_mae))
print("Cross-Validation Mean Squared Error (MSE): {:.2f}".format(mean_cv_mse))
print("Cross-Validation Coefficient of Determination (R^2): {:.2f}".format(mean_cv_

# Predict on training set
y_train_pred = best_rf.predict(X_train)

# Model evaluation metrics on training set
mae_train = mean_absolute_error(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

print("Training Set - Mean Absolute Error (MAE):", mae_train)
print("Training Set - Mean Squared Error (MSE):", mse_train)
print("Training Set - Coefficient of Determination (R^2):", r2_train)

# Plot learning curve
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
```

```
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

# Plot learning curve
plot_learning_curve(best_rf, "Learning Curve for Random Forest", X_train, y_train,
plt.show()

# Predict on test set
y_test_pred = best_rf.predict(X_test)

# Model evaluation metrics on test set
mae_test = mean_absolute_error(y_test, y_test_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("Test Set - Mean Absolute Error (MAE):", mae_test)
print("Test Set - Mean Squared Error (MSE):", mse_test)
print("Test Set - Coefficient of Determination (R^2):", r2_test)
```

```
Best Parameters of Random Forest: {'bootstrap': True, 'max_depth': 10, 'max_feature
s': 'sqrt', 'min_samples_leaf': 8, 'min_samples_split': 2, 'n_estimators': 300}
Validation Set - Mean Absolute Error (MAE): 40544.26499983374
Validation Set - Mean Squared Error (MSE): 2569572761.1546893
Validation Set - Coefficient of Determination (R^2): 0.511277139105486
Cross-Validation Mean Absolute Error (MAE): 39969.71
Cross-Validation Mean Squared Error (MSE): 2455773573.58
Cross-Validation Coefficient of Determination (R^2): 0.57
Training Set - Mean Absolute Error (MAE): 36551.86368450629
Training Set - Mean Squared Error (MSE): 2062609152.726223
Training Set - Coefficient of Determination (R^2): 0.6374741251309597
```



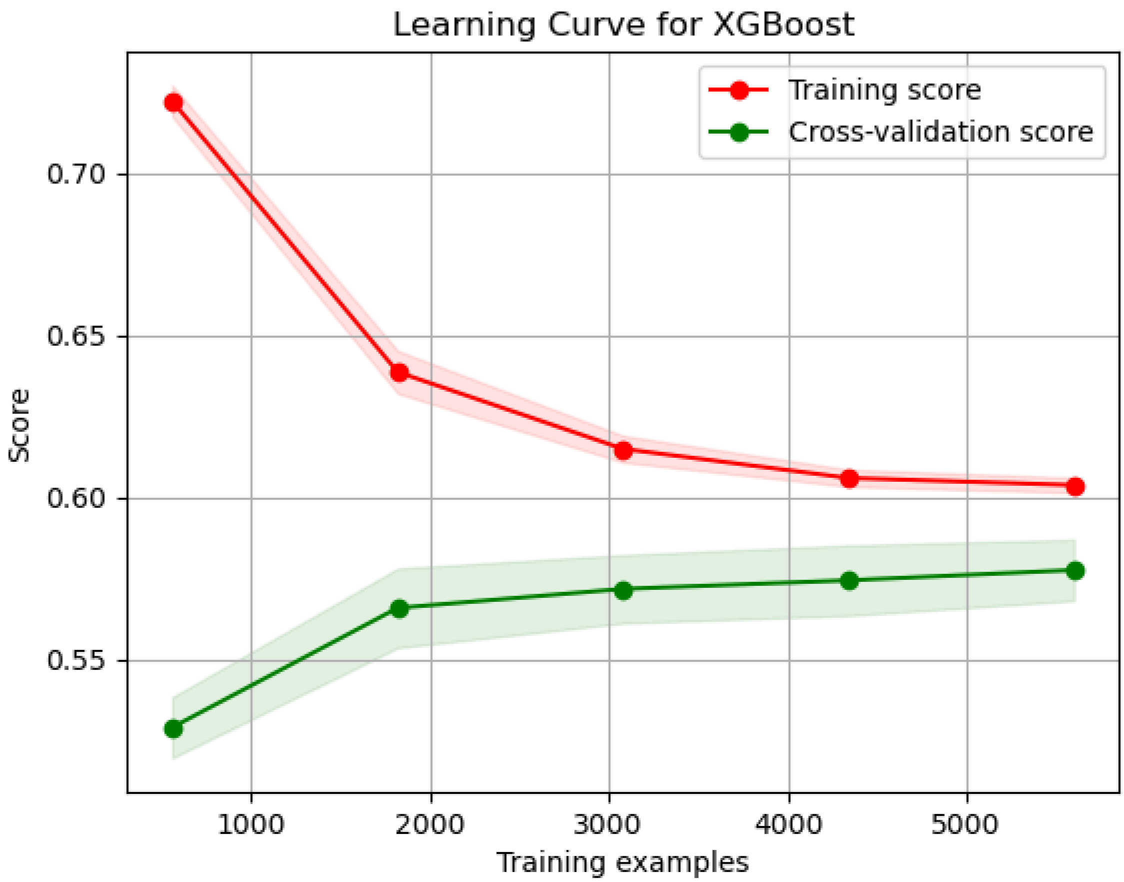

```
Test Set - Mean Absolute Error (MAE): 37775.517369300345
Test Set - Mean Squared Error (MSE): 2255574714.1439543
Test Set - Coefficient of Determination (R^2): 0.5934353428284687
```

In [22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, learning
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler

# Load dataset
data_path = "C:\\Users\\USER\\sampled_cleaned_data.csv"
df = pd.read_csv(data_path)

# Define features and target variable
X = df.drop('Price', axis=1)
y = df['Price']

# Split dataset into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

```
X_test_scaled = scaler.transform(X_test)

# Define hyperparameter grid
param_grid = {
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200, 300],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.3],
    'reg_alpha': [0, 0.1, 0.3],
    'reg_lambda': [0, 0.1, 0.3]
}

# Define k-fold cross-validation
kfold = KFold(n_splits=5)

# Instantiate XGBoost regressor
xgb = XGBRegressor()

# Instantiate GridSearchCV
grid_search = GridSearchCV(xgb, param_grid, cv=kfold, scoring='neg_mean_squared_err

# Fit grid search to the data
grid_search.fit(X_train_scaled, y_train)

# Get best parameters
best_params = grid_search.best_params_
best_xgb = grid_search.best_estimator_

# Model evaluation metrics on validation set
y_val_pred = best_xgb.predict(X_val_scaled)
mae_val = mean_absolute_error(y_val, y_val_pred)
mse_val = mean_squared_error(y_val, y_val_pred)
r2_val = r2_score(y_val, y_val_pred)

print("Best Parameters of XGBoost:", best_params)
print("Validation Set - Mean Absolute Error (MAE):", mae_val)
print("Validation Set - Mean Squared Error (MSE):", mse_val)
print("Validation Set - Coefficient of Determination (R^2):", r2_val)

# Cross-validation performance metrics
cv_mae_scores = []
cv_mse_scores = []
cv_r2_scores = []
for train_index, val_index in kfold.split(X_train_scaled):
    X_train_fold, X_val_fold = X_train_scaled[train_index], X_train_scaled[val_inde
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]
    best_xgb.fit(X_train_fold, y_train_fold)
    y_val_pred_fold = best_xgb.predict(X_val_fold)
    fold_mae = mean_absolute_error(y_val_fold, y_val_pred_fold)
    fold_mse = mean_squared_error(y_val_fold, y_val_pred_fold)
    fold_r2 = r2_score(y_val_fold, y_val_pred_fold)
    cv_mae_scores.append(fold_mae)
    cv_mse_scores.append(fold_mse)
    cv_r2_scores.append(fold_r2)
```

```
mean_cv_mae = np.mean(cv_mae_scores)
mean_cv_mse = np.mean(cv_mse_scores)
mean_cv_r2 = np.mean(cv_r2_scores)
print("Cross-Validation Mean Absolute Error (MAE): {:.2f}".format(mean_cv_mae))
print("Cross-Validation Mean Squared Error (MSE): {:.2f}".format(mean_cv_mse))
print("Cross-Validation Coefficient of Determination (R^2): {:.2f}".format(mean_cv_

# Model evaluation metrics on training set
y_train_pred = best_xgb.predict(X_train_scaled)
mae_train = mean_absolute_error(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

print("Training Set - Mean Absolute Error (MAE):", mae_train)
print("Training Set - Mean Squared Error (MSE):", mse_train)
print("Training Set - Coefficient of Determination (R^2):", r2_train)

# Plot learning curve
def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=None, train_sizes=n
    plt.figure()
    plt.title(title)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

plot_learning_curve(best_xgb, "Learning Curve for XGBoost", X_train_scaled, y_train
plt.show()

# Model evaluation metrics on test set
y_test_pred = best_xgb.predict(X_test_scaled)
mae_test = mean_absolute_error(y_test, y_test_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("Test Set - Mean Absolute Error (MAE):", mae_test)
print("Test Set - Mean Squared Error (MSE):", mse_test)
print("Test Set - Coefficient of Determination (R^2):", r2_test)
```

```
Best Parameters of XGBoost: {'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate':
0.1, 'max_depth': 3, 'n_estimators': 100, 'reg_alpha': 0.3, 'reg_lambda': 0, 'subsam
ple': 1.0}
Validation Set - Mean Absolute Error (MAE): 39946.71830208333
Validation Set - Mean Squared Error (MSE): 2500849381.107564
Validation Set - Coefficient of Determination (R^2): 0.5243480617953262
Cross-Validation Mean Absolute Error (MAE): 39554.38
Cross-Validation Mean Squared Error (MSE): 2403530542.18
Cross-Validation Coefficient of Determination (R^2): 0.58
Training Set - Mean Absolute Error (MAE): 38574.38163950893
Training Set - Mean Squared Error (MSE): 2284883603.268745
Training Set - Coefficient of Determination (R^2): 0.5984069855628755
```




```
Test Set - Mean Absolute Error (MAE): 37392.866765625
Test Set - Mean Squared Error (MSE): 2211439098.7195115
Test Set - Coefficient of Determination (R^2): 0.6013907349693596
```

In [ ]: