In [1]:
```
#Impoting libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import os
```

In [2]:
```
#reading_inputs
files= [file for file in os.listdir(r'C:\Users\Rahul\Downloads\house_price_predi
print(files)
os.chdir(r'C:\Users\rahul\Downloads\house_price_prediction')
df=pd.read_csv('house_price_2nd_data.csv')
```

```
['.ipynb_checkpoints', 'Cell 12.csv', 'Code.ipynb', 'Code_in_PDF.pdf', 'House Pri
ce Prediction Dataset.csv', 'house_price_2nd_data.csv']
```

In [3]:
```
#check there is no null value in the data
print(df.isnull().sum())
```

```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

In [4]:
```
#name of the columns
print(df.columns)
df.head()
```

```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'street', 'city',
       'statezip', 'country'],
      dtype='object')
```

Out[4]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	0

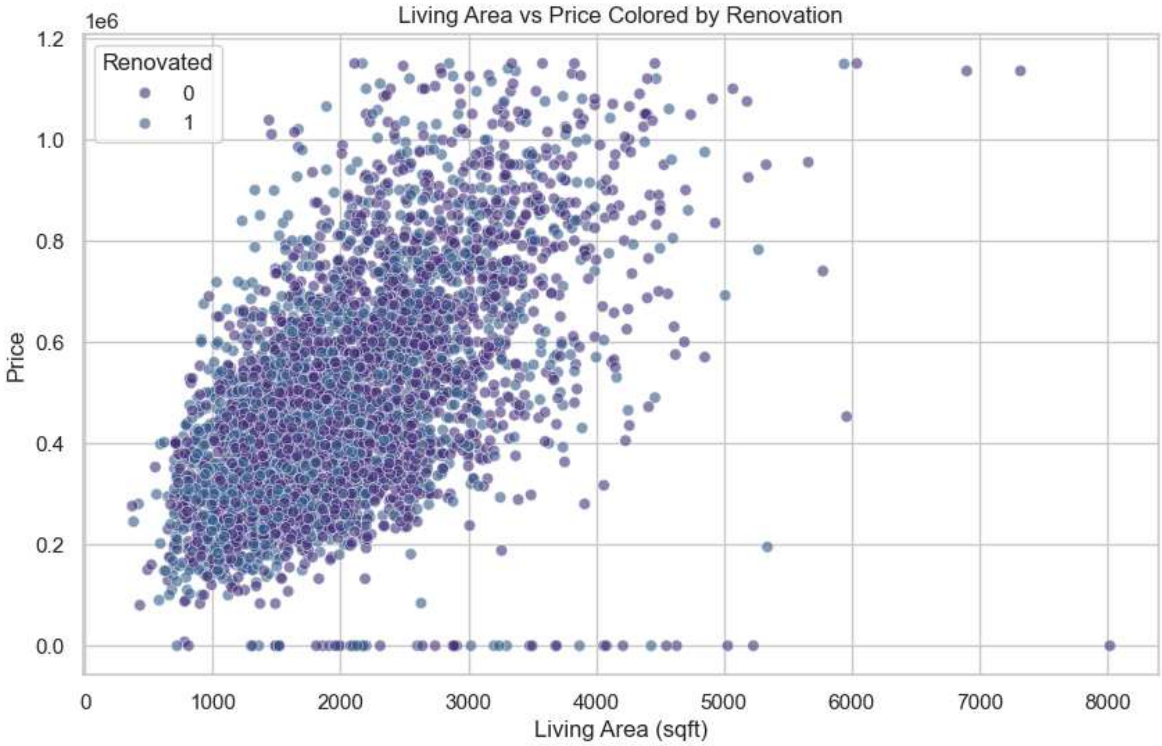
In [5]:
```
#chaning columns header if there is blanks, and convrting the header names to lo
df.columns=df.columns.str.strip().str.lower().str.replace(" ","_")
```

In [6]:
```
#changing numeric_columns to float
numeric_cols = ['price', 'sqft_living', 'sqft_lot', 'sqft_above',
                'sqft_basement']
for col in numeric_cols:
    df[col]=df[col].astype(float)
```

In [7]:
```
#based on year calculating age of house
df['age_of_house']=2025-df['yr_built']
```

In [8]:
```
#if house was renovated then what is the age age after renovation , otherwise ta
df['final_age']=0
for i in range(len('age_of_house')):
    if (df['yr_renovated'][i]==0):
        df['final_age'][i]=df['age_of_house'][i]
    else:
        df['final_age'][i]=2025-df['yr_renovated'][i]
df['is_renovated']=0
df['is_renovated']=df['yr_renovated'].apply(lambda x:1 if x!=0 else 0)
```

In [35]:
```
# Square footage vs price
sns.set(style="whitegrid", palette="viridis")
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='sqft_living', y='price', hue='is_renovated', alpha=0
plt.title("Living Area vs Price Colored by Renovation")
plt.xlabel("Living Area (sqft)")
plt.ylabel("Price")
plt.legend(title="Renovated")
plt.show()
#clearly visible that Living are affects the price positively(positive correlati
```

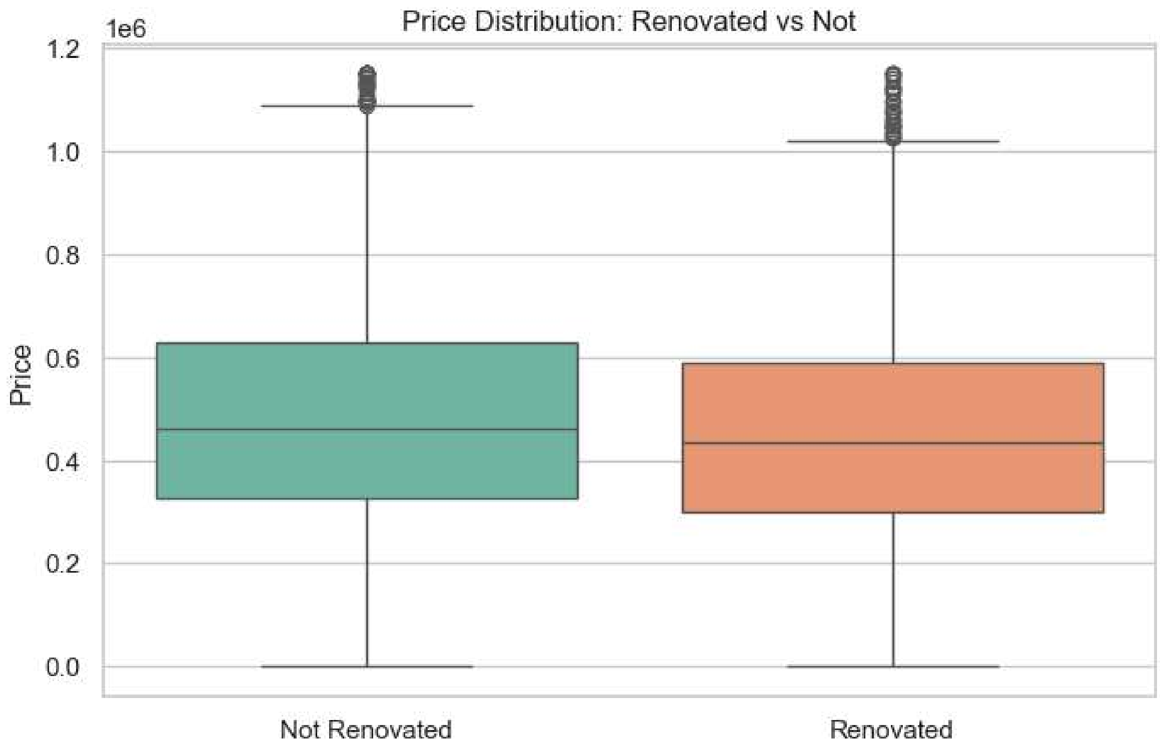


In [38]:
```
#impact of renovation on Price
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='is_renovated', y='price', palette='Set2')
plt.title("Price Distribution: Renovated vs Not")
plt.xticks([0, 1], ['Not Renovated', 'Renovated'])
#plt.xlabel("")
plt.ylabel("Price")
plt.show()
```

```
C:\Users\amrah\AppData\Local\Temp\ipykernel_19208\2524332779.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(data=df, x='is_renovated', y='price', palette='Set2')
```



In [9]:
```
#checking Response variable(#Y variable i.e price)
plt.figure(figsize=(8, 4))
sns.histplot(df["price"], bins=50, kde=True)
plt.title("Price Distribution")
plt.xlabel("Price")
plt.ylabel("Frequency")
plt.show()
```

In [10]:
```
#Checking for outliers in Response variable by boxplot
import matplotlib.pyplot as plt
import seaborn as sns

# Set plot style
sns.set(style="whitegrid")

# Create the boxplot
```

```
plt.figure(figsize=(6, 4))
sns.boxplot(y=df["price"])
plt.title("Boxplot of House Prices")
plt.ylabel("Price")
plt.tight_layout()
plt.show()
```

In [11]:
```
#outlier detection from ressponse (y)
Q1=df['price'].quantile(0.25)
Q3=df['price'].quantile(0.75)
IQR=Q3-Q1
lower_bound=Q1-1.5*IQR
upper_bound=Q3+1.5*IQR
```

In [12]:
```
#Outlier removal
df_clean = df[(df["price"] >= lower_bound) & (df["price"] <= upper_bound)].reset

print(f"Original rows: {len(df)}")
print(f"After removing Price outliers: {len(df_clean)}")
```

```
Original rows: 4600
After removing Price outliers: 4360
```

In [13]:
```
df=df_clean
```

In [14]:
```
#There are 44 cities, Dummy variables can be too much in mumber if we create 43
city_df=df[['city']].groupby(['city']).agg({'city':'count'})
```

In [15]:
```
city_df.rename(columns={'city': 'city_count'}, inplace=True)
```

In [16]:
```
city_df=city_df.reset_index()
```

In [17]:
```
city_df = city_df.sort_values('city_count', ascending=False)
```

In [18]:
```
city_df=city_df.head(10)
```

In [19]:
```
city_list=city_df['city'].to_list()
```

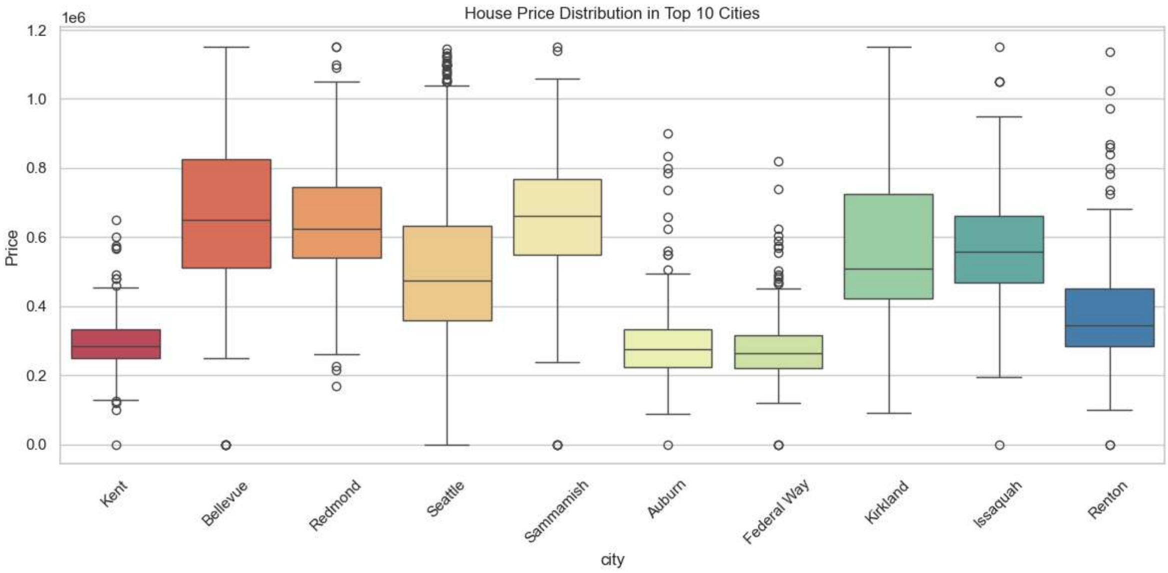
In [20]:
```
df['city']=df['city'].apply(lambda x: x if x in city_list else 'other')
```

In [41]:
```
#City wise House price
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[df['city'].isin(city_list)], x='city', y='price', palette='S
plt.title("House Price Distribution in Top 10 Cities")
plt.xticks(rotation=45)
plt.ylabel("Price")
plt.tight_layout()
plt.show()
```

```
C:\Users\amrah\AppData\Local\Temp\ipykernel_19208\1326741434.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(data=df[df['city'].isin(city_list)], x='city', y='price', palette
='Spectral')
```



In [21]:
```
df.reset_index(inplace=True)
```

In [43]:
```
#Checking Correlation of the Variables by Heatmap(for nueric Columns only)
plt.figure(figsize=(12, 10))
sns.heatmap(df[numeric_cols + ['price']].corr(), annot=True, cmap='coolwarm', ce
plt.title("Correlation Matrix")
plt.show()
```

In [22]:
```
#Making Regression inputs ready

x=df.drop(['price','date','yr_built','yr_renovated','street','statezip','country
```

In [23]:
```
#assigning dummy variable for city
df_encoded=pd.get_dummies(df['city'],drop_first=True).astype(int)
```

In [24]:
```
x=pd.concat([x,df_encoded],axis=1)
```

In [25]:
```
x.drop(['index'],axis=1,inplace=True)
```

In [26]:
```
#making response variable ready for regression
y=df['price']
```

In [27]:
```
#starting calculations for PCA, startig with
x_numeric=x
#  standardize data
x_mean=np.mean(x_numeric,axis=0)
x_std=np.std(x_numeric,axis=0)
x_numeric=(x_numeric-x_mean)/x_std
```

In [28]:
```
#calculating COV matrix on standardized data
cov_matrix=np.cov(x_numeric.T)
```

In [29]:
```
# Compute eigenvalues and eigenvectors
eigen_values, eigen_vectors = np.linalg.eigh(cov_matrix)

# Sort them in descending order of eigenvalue
sorted_indices = np.argsort(eigen_values)[::-1]
eigen_values = eigen_values[sorted_indices]
eigen_vectors = eigen_vectors[:, sorted_indices]
```

In [30]:
```
# Choose number of components (e.g., enough to explain 95% variance)
total_variance = np.sum(eigen_values)
explained_variance_ratio = eigen_values / total_variance
cumulative_variance = np.cumsum(explained_variance_ratio)

# Pick number of components that explain >= 95% variance
k = np.argmax(cumulative_variance >= 0.95) + 1
print(f"Number of components explaining 95% variance: {k}")
```

```
Number of components explaining 95% variance: 18
```

In [31]:
```
# Select top-k eigenvectors
principal_components = eigen_vectors[:, :k]

# Project data
x_pca = np.dot(x_numeric, principal_components)

#PCA is done till here .
```

In [32]:
```
#Running linear regression on this transformed data
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Train/test split on reduced data
X_train, X_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.3, ran

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R² Score:", r2_score(y_test, y_pred))
```

```
RMSE: 153146.16018019523
R² Score: 0.5175706832556513
```

In [33]:
```
adjusted_r2 = 1 - (1 - r2_score(y_test, y_pred)) * (len(y_test) - 1) / (len(y_te
```

In [34]:
```
adjusted_r2
```

Out[34]:
```
0.5108338890730304
```

In [ ]: